SOLVE2000:

Some Proposed Changes In Solve

John M. Gipson

USNO

December 4, 1998

Some Current Problems in Solve:

1. Hard to understand.
2. Hard to maintain.
3. Hard to extend.
4. Hard to use.
5. Slow(er than it needs to be.)

Primary Cause of Problems:

1. Child of many parents
2. Computer limitations at time written.
3. Memory expensive, size of programs limited.
4. Operating systems changed over time.
5. Compilers changed over time.
6. Computers changed over time.
7. Problems changed over time.

Secondary Causes of Problems

1. Inconsistent programming styles, naming conventions, etc.
2. Obsolete code. Much of the code solves (programming and other) problems that no longer exist.
3. Archaic data structures, e.g., Holleriths.
4. Unnessarily short names, subroutines. No need to restrict to 6 characters. (Example: What does *depar* do?)
5. Extensive use of bit arrays to save space.
6. Obscure code. (e.g., how do you tell which parameters are arc, which global?
7. Implicit data structure. Data structure hard wired into solve.
8. Too many programs because of initial size limitations (ADDER/XDDER).
9. Too few subroutines. Unnecesary duplication of code. Leads to inconsistencies, hard to maintain.

All this leads to current situation in solve:

1. Hard to understand.
2. Hard to maintain.
3. Hard to extend.
4. Hard to use.
5. Slow(er than it needs to be.)

Examples:

1. Obscure code: How do you find out which parameters are arc parameters? A.) Find all parameters. B.) Turn off arc parameters. C.) Find all parameters. D.) Compare results of A& B.
2. Hard to extend: Difficulty of incorporating piecewise linear gradients.
3. Hard to extend: Difficulty of combining VLBI& GPS data in solve. Easy to do outside of solve. After 3 months of effort, task abandoned because bookkeeping became to onerous.

Some (Not All) Solutions

1. Better coding standards.
2. Different data structure.
3. Alternate fast matrix inversion.

Better Coding in Solve

Documentation of algorithms.

Documentation of mathematical algorithms used in solve. L. Petrov and myself regularly produced memos on what we did in solve. Such memos should be gathered in a central place, and perhaps placed on the web.

References in the code referring to external documentation. This is done regularly in CALC.

Documentation of programs.

External documentation about what each module in solve does.

Program flow is poorly documented, somewhat of a mystery to even expert users. Currently no flowchart of all the modules. Such a flow chart would aid in debugging and understanding.

Subroutine documentation and library.

Proposal: Have a central reposity which describes what all the subroutines do in solve, and their calling conventions. This does not exist. As a consequence, each person has their own collection of favorite routines. I frequently write routines on the fly because I don't know what is available.

Style Manual

Consistency improves maintainability, and is something to strive for.

Everyone has their own coding style, which is more or less consistent. There is currently no attempt to enforce/encourage consistency between programmers.

Proposal: Style manual which specifies variable naming conventions, and formatting conventions. One possibility: adopt Hungarian notation widely used in C programming.

Purge the code of obsolete constructs or usage, such as Holleriths.

Solve Data Structure

Solve knows deep in the code that you are dealing with VLBI data. This makes it very difficult to combine VLBI data with other data types (GPS, SLR).

Because of memory limitations at the time the code was written, much of the data structure is determined by bit arrays. These are hard for a human to decode.

Currently the kind and order of partials is hard wired in solve. In global solutions this leads to reordering as new parameters are added, which is costly in terms of time.

In a word, most of the data structure is implicit.

Proposal: New *explicit* data structure.

Specify all solve parameters completely by an ascii string:

Geo. Part. | Time Part. | Epoch | Start Time | End Time | Aux Info

Examples:

```
Gilcreek Atm  |Pwise Linear|9401121800|94011121700|9401121930|

Gilcreek Atm  |Diurnal     |9401121800|94011121700|9401121930|

Kokee    U Pos|Diurnal     |9401121800|94011121700|9401121930|

UT1           |Poly 0      |9401121800|94011121700|9401121930|

UT1           |Poly 1      |9401121800|94011121700|9401121930|

UT1           |Tidal       |9401121800|94011121700|9401121930|0 0 0 0 0 -1
```

Motivation:

Natural. This representation mimics the functional form of the partials:

$$Partial = Geom\_partial \times Time\_partial$$

Advantages of new data structure.

1. Makes explicit the partial. Current scheme hides this information. When this information is required, which it often is, you must call *get_names*. (At GSFC *parms* was replaced by routine *get_names* in 1996. *get_names* returns ascii strings instead of holleriths but is otherwise the same as *parms*.)

2. Bookkeeping derived from partial names. Currently several parts of the code need to know the order of the partials to do their bookkeeping correctly. Hence if you add new partials, need to make changes in several places.

3. No duplication of code. The new scheme would automatically use same code for atmosphere/clocks/eop time partials.

4. Automatically ensures consistency of code. Fix a bug once, instead of many times.

5. Easy to modify time behavior of partial. For example, to estimate piece-wise linear gradients, all you need to do is generate the appropriate ascii

strings. Same would go for piecewise linear station positions.

6. Data combination easier since all the information about the partial is in one place. Don't need to force solve to think that GPS or SLR data is really VLBI data.

7. No distinction between global and arc parameters since the parameter list specifies the time interval. Makes it easier to force continuity across day changes.

8. Data combination with VLBI, SLR is easier, since all the information about the partials is in one place.

9. Contribution of observations to most partials is only for a limited time.
   - Don't have to calculate null-partials. (From parameter list know which data contributes to which partial.)
   - Don't have to update normal equations with 0.
   - Can squeeze out partials when we are done with them.

Alternative Fast Matrix Inversion

L. Petrov's scheme relies on intimate knowledge of structure of normal equations. It knows that we estimate (only) clocks, atmospheres and EOP as piecewise linear functions. It also knows where these occur in the parameter list.

I propose a new scheme. Called "Ordered Matrix Inversion" or OMI. Besides the normal matrices *N* and *A*, this uses as input two auxiliary arrays:

tbeg: Time when a parameter turns on.

tend: Time when a parameter turns off.

Broadly speaking, this scheme inverts the normal equations in time order, squeezing out parameters as they are turned off.

In the simplest form, this scheme can be applied to invert the normal equations of a single arc, or do the necessary inversion for arcpe.

A more ambitious proposal would use this scheme to do all of the matrix inversion in solve.

Ordered Matrix Inversion

Consider the normal equations for a particular arc. We want to squeeze out parameters which are no longer on. Let these parameters which we want to squeeze out be denoted by *"a"* where the a stands for current arc. Label the parameters which the *a* couple to by *"g"* (=current global). The remaining parameters are labeled by *"r"*. The normal equations are:

$$\begin{pmatrix} N_{gg} & N_{gr} & N_{ga} \\ N_{rg} & N_{rr} & 0 \\ N_{ag} & 0 & N_{aa} \end{pmatrix} \begin{pmatrix} A_g \\ A_r \\ A_a \end{pmatrix} = \begin{pmatrix} B_g \\ B_r \\ B_a \end{pmatrix}$$

Where each of the N's are matrices, and the A's and B's are vectors. We can "squeeze out" the arc parameters. The reduced normal equations are:

$$\begin{pmatrix} N_{gg} - N_{ga}N_{aa}^{-1}N_{ag} & N_{gr} \\ N_{rg} & N_{rr} \end{pmatrix} \begin{pmatrix} A_g \\ A_r \end{pmatrix} = \begin{pmatrix} B_g - [N_{ga}N_{aa}^{-1}]B_a \\ B_r \end{pmatrix}$$

The solution for the arc parameters is:

$$A_a = N_{aa}^{-1}B_a - [N_{aa}^{-1}N_{ag}]A_g$$

The covariance matrix of the full system can be found from the covariance matrix of the reduced system.

Let

$$Cov(g,r) \equiv \begin{pmatrix} N_{gg} - N_{ga}N_{aa}^{-1}N_{ag} & N_{gr} \\ N_{rg} & N_{rr} \end{pmatrix}^{-1} = \begin{pmatrix} M_{gg} & M_{gr} \\ M_{rg} & M_{rr} \end{pmatrix}$$

Then:

$$Cov(g,r,a) = \begin{pmatrix} M_{gg} & M_{gr} & -M_{gg}[N_{ga}N_{aa}^{-1}] \\ M_{rg} & M_{rr} & -M_{gr}[N_{ga}N_{aa}^{-1}] \\ \ldots & \ldots & N_{aa}^{-1} + [N_{aa}^{-1}N_{ga}]M_{gg}[N_{ga}N_{aa}^{-1}] \end{pmatrix}$$

Proof: Left to the reader.

Can build up the covariance matrix step by step (if it is desired).

Comments:

1. To calculate the reduced normal equations need to calculate N_{aa}^{-1} and $N_{ga}N_{aa}^{-1}$

2. The matrices in [...] are related by transposition.

3. This process can be repeated again.

4. At the very end of the forward direction we have a solution for the remaining current-global parameters, and the associated covariance. These parameters all parameters which were on at the end of the experiment. Frequently this is all we want.

5. If we want to, we can find the solution for the arc parameters one level up without any further matrix inversion.

6. Each step can be done in place:

$$\begin{aligned} N_{aa} &\leftarrow N_{aa}^{-1} \\ B_a &\leftarrow N_{aa}^{-1}B_a \\ N_{ga} &\leftarrow N_{ga}N_{aa}^{-1} \end{aligned}$$

General scheme for fast matrix inversion in solve.

1. Proceed in time order, starting at earliest time, and proceeding to last time.

2. Build up normal equation in time order. This uses the tbeg array.

3. As parameters turn off, squeeze them out. This uses tend array.

4. At end of forward solution, are left with "global" parameters, and parameters which were on at the end of the experiment. Invert reduced normal equations.

5. If we want rest of parameters, do back solution.

If we want covariance matrix of rest of parameters, also do back solution for these.

Timing Considerations

For simplicity assume that we have N_{arc} number of "arcs", and each arc involvesN_A parameters. Assume each arc couples only to its neighbors. Also assume we have N_G parameters which are on for the whole experiment. The total number of parameters is:

$$N_{Tot} = N_G + N_{arc} \times N_A$$

The time it takes standard solve to invert the normal equations goes like:

$$T_{solve} \sim N_{Tot}^3/2$$

One can show that the time it takes for the forward solution in the proposed scheme is:

$$T_{Forward} \sim N_{Tot} \times (2N_A + N_G)^2/2$$

Hence the ratio is:

$$T_{Forward}/T_{solve} = \left(\frac{2N_A+N_G}{N_T}\right)^2 \approx \left(\frac{2}{N_{arc}}\right)^2$$

This ignores overhead, and implies speedups of upto 100 or more, which is unrealistic. But experimental tests indicate that the speedup is substantial.

For many purposes you don't need the back solution with the full covariance matrix. If you need this, then you can show that

$$T_{Back}/T_{solve} = \left(\frac{2N_A+N_G}{N_T}\right) \approx \left(\frac{2}{N_{arc}}\right)$$

This is much faster than standard solve.

Does this proposed algorithm work?

Yes

I have implemented this algorithm in both norml and arcpe. Source is in:

leo://data18/mk3/src/solve/norml_fast_jmg

leo://data18/mk3/src/solve/arcpe_fast_jmg

There are several modes of running, depending on whether you want full covariance, approximate covariance, or minimal covariance (only the parameters on at the end of the experiment.)

For one of the CONT94 experiments, the speedup of the matrix inversion for the combined solution (forward and back) was a factor of 8 faster than standard solve. Namely 2 seconds vs 16 secs.

Speed is comparable to L. Petrov's technique.

No attempt has been made to speed up internal code, e.g., matrix multiplications could be done using vector instruction set.

tb, te arrays made by calling *get_names*, and then finding on and off times by looking at parameter names. This leads to correct results in 90% of the cases, but leads to errors in some cases because of inconsistencies in converting from Julian date to Year, month, hour,minute second. Because of this I had to include checks to make sure the results were correct. This also slows down the algorithm.

Code spends a fair amount of time reordering data at the end. This could be eliminated if we went with "explicit parameter labeling". All that would need to be done in this case would be to re-order the parameter list.

Advantages of OMI Algorithm

1. Logic is simple and direct.
2. Requires minimimal modifications to solve. (Just need time arrays.)
3. All bookkeeping done within routine.
4. Easily adapted to do standard arcpe.
5. No restrictions on functional form of partials, i.e., piecewise linear with commensurate rate breaks for clocks, atmospheres and EOP.
6. If we estimated new piecewise linear functions, would handle it automatically.

Further extensions.

1. Reading and using sinex files/geodyn files.
2. Unification of global/arc parameters. Once a parameter is done, it is squeezed out. Easier to have multi-arc parameters.
3. Ability to turn off/on parameter estimation w/o re-generating normal equations. For example, baseline and TRF solutions could be generated from a single CGM. This would ensure consistency.
4. Ability to use Kalman filter for some or all of the parameters.